UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/783,598	02/20/2004	David Wortendyke	MS1-2021US	1564

22801 7590 03/25/2008

LEE & HAYES PLLC
421 W RIVERSIDE AVENUE SUITE 500
SPOKANE, WA 99201

EXAMINER	
CHILES, AARON T	
ART UNIT	PAPER NUMBER
2169	
MAIL DATE	DELIVERY MODE
03/25/2008	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	Application No.	Applicant(s)	
	10/783,598	WORTENDYKE ET AL.	
	Examiner	Art Unit	
	AARON CHILES	2169	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE *3* MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *13 December 2007*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-5,7-15,17-21,23-28,30,31,33,35-37 and 39-48* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-5,7-15,17-21,23-28,30,31,33,35-37 and 39-48* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☒ The specification is objected to by the Examiner.

10)☒ The drawing(s) filed on *20 February 2004* is/are: a)☐ accepted or b)☒ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☐ Notice of References Cited (PTO-892)
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.
4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
5) ☐ Notice of Informal Patent Application
6) ☐ Other: _____.

DETAILED ACTION

Status of claims:

Claims 1-5, 7-15, 17-21, 23-28, 30-31, 33, 35-37, 39-48 are pending in this office action.

Claims 1, 8, 10, 17-19, 23-24, 26, 31, 37, 39 are amended.

Claims 6, 16, 22, 29, 32, 34, 38 are cancelled by applicant request.

Claims 44-48 are newly added.

Response to Arguments

1. Applicant's arguments with respect to claims 1-43 have been considered but are moot in view of the new ground(s) of rejection.

Drawings

2. New corrected drawings in compliance with 37 CFR 1.121(d) are required in this application because of previously explained objections, see prior action. Applicant is advised to employ the services of a competent patent draftsperson outside the Office, as the U.S. Patent and Trademark Office no longer prepares new drawings. The corrected drawings are required in reply to the Office action to avoid abandonment of the application. The requirement for corrected drawings *will not be held in abeyance*.

Specification

3. The objections to the specification will be held in abeyance in response to the request by the applicant.

4. The specification is objected to as failing to provide proper antecedent basis for the claimed subject matter. See 37 CFR 1.75(d)(1) and MPEP § 608.01(o). Correction of the following is required: The term 'physical' in claims 10, 19, 26, and 37, do not appear in the specification.

Claim Objections

5. Claims 33 and 35 depend from a canceled claim. There are herein evaluated as if depending from claim 31, as claim 32 has been cancelled, and its limitations moved to claim 31.

Claim Rejections - 35 USC § 103

6. The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this Office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

7. **Claims 1-4, 7-13, 15, 17, 37, 39, 41 are rejected under 35 U.S.C. 103(a) as being unpatentable over Aguilera et al. (NPL #1, Matching Events in a Content-**

based Subscription System), in view of Graefe et al. (NPL #2, Dynamic Query Evaluation Plans).

As per claim 1, **Aguilera et al.** discloses a method for updating a filter engine opcode tree, comprising the following steps:

(a) compiling a new query to derive a series of opcode objects (Sec. 3, Para. 2, "each subscription is a conjunction of *elementary predicates*);

(b) traversing the opcode tree according to the series of opcode objects until an opcode object is encountered that is not included in the opcode tree, opcode objects being represented in the opcode tree as opcode nodes (App. A, Para. 2, lines 5-7);

(c) adding new opcode tree opcode nodes to correspond to the encountered opcode object and subsequent opcode objects in the series of opcode objects (App. A, Para. 2, lines 7-8).

However, **Aguileraet al.** does not explicitly disclose

(d) updating a branch node in the opcode tree to add a reference to the new opcode nodes, the branch node being referenced from a parent opcode node that corresponds to a last opcode object from the series of opcode objects that was found in the traversal of the opcode tree; and

(e) implementing an optimized branch node that includes an optimized indexed lookup procedure, wherein the indexed lookup procedure is configured to return a set of (key, value) pairs, a single (key, value) pair corresponding to one branch.

However, **Graefe et al.** discloses

(d) updating a branch node in the opcode tree to add a reference to the new opcode nodes, the branch node being referenced from a parent opcode node that corresponds to a last opcode object from the series of opcode objects that was found in the traversal of the opcode tree (Pg. 362, second full paragraph, "this operator provides the same *open, next, close* protocol as the other operators and can therefore be inserted into a query plan at any location." This, also, shows that the 'choose plan operator' would be its own opcode node, when used in conjunction with **Aguilera et al.** Additionally, pg. 361, last paragraph of section 5, "If there is no gain in using a dynamic access module, the decision tree can by an empty function");

(e) implementing an optimized branch node that includes an optimized indexed lookup procedure, wherein the indexed lookup procedure is configured to return a set of (key, value) pairs, a single (key, value) pair corresponding to one branch. (This limitation is nothing more than a recitation of the steps taken by a hash algorithm, or related types of indexing methods, Applicant's spec. pg. 22, lines 3-7. As such, it would have been obvious to one of ordinary skill in the art at the time of the invention to use an efficient indexing method, when appropriate. For instance, **Graefe et al.** page 360 "using a hash index").

It would have been obvious to one of ordinary skill in the art at the time of the invention to evaluation system of **Aguilera et al.** with the choose plan operators of **Graefe et al.** in order to increase its efficiency as described in **Graefe et al.**

As per claim 2, **Aguilera et al.** as modified, also discloses wherein one or more of the steps are performed dynamically at runtime (**Aguilera et al.**, pg. 2, Col. 1, lines 22-24). It is a well-known goal of subscription server systems, as disclosed by **Aguilera et al.**, to maximize availability. As such, it is anticipated by **Aguilera et al.** that the incremental updates would be done at run-time.

As per claim 3, **Aguilera et al.** as modified, also discloses further comprising performing steps (b) and (c) in a component of the filter engine (**Aguilera et al.,** code listing, pg. 9, the pre-processing steps are a procedure contained within the tree matching algorithm, and as such, do not pertain to a stand alone program, and are therefore inherently part of the tree matching algorithm (filter engine)).

As per claim 4, **Aguilera et al.** as modified, also discloses further comprising executing the opcode tree against an input to evaluate the new query and one or more other queries against the input (**Aguilera et al.**, Sec. 2, Par. 2, lines 1-7).

As per claim 7, **Aguilera et al.** as modified, discloses the branch node further comprising updating the branch node to include an indexed lookup routine that references several dependent opcode nodes that perform a similar function (**Aguilera et al.**, Section 5, dynamic query evaluation plans include a decision procedure which is used at initialization time, and intermittently thereafter to choose the optimal evaluation strategy).

As per claim 8, **Aguilera et al.** as modified, discloses further comprising analyzing opcode nodes that depend from the branch node and including the indexed lookup routine only if including the indexed lookup routine provides more efficient processing of the dependent nodes that a generic branch node processing routine (**Graefe et al.** Section 5, dynamic query evaluation plans include a decision procedure which is used at initialization time, and intermittently thereafter to choose the optimal evaluation strategy).

As per claim 9, **Aguilera et al.** as modified, does not explicitly disclose the indexed lookup routine further comprising one of the following routines: a hash routine; a routine that uses tries; an interval tree routine. However, as **Aguilera et al.** is silent as to which indexing method to use, it would obvious to one of ordinary skill to choose an efficient indexing method based on the attribute in question, which includes a hash routine; a routine that [or], an interval tree routine. For instance, see **Graefe et al.** on page 360 “using a hash index”.

As per claim 10, **Aguilera et al.** discloses a filter engine (Section 3, tree matching algorithm) stored on one or more computer-readable media, in a physical embodiment, comprising:

a filter table that includes a plurality of queries, at least two of the queries including a common prefix (Sec. 3, matching tree);

a compiler configured to compile each query into a series of opcode blocks (App. A, pre-processor);

an opcode tree stored in memory and including opcode nodes that each correspond to an opcode block such that executing the opcode nodes evaluates the plurality of queries (pg. 2, col 1, lines 29-35), at least one opcode node corresponding to an opcode block included in the common prefix (Appendix A);

an opcode merger configured to merge a new query to the opcode tree by adding at least one opcode node that corresponds to the new query to the opcode tree (pg. 54, col. 1, lines 22-24);

However, **Aguilera et al.** does not explicitly disclose

wherein, when an opcode node will depend from a branch node when added to the opcode tree, identifying one or more child opcode nodes that depend from the branch opcode, and

implementing an optimized branch node that includes an optimized indexed lookup procedure if such implementation would increase branch processing efficiency and referencing the opcode node from the optimized branch node,

wherein the indexed lookup procedure is configured to return a set of (key, value) pairs, a single (key, value) pair corresponding to one branch.

However, **Graefe et al.** discloses wherein, when an opcode node will depend from a branch node when added to the opcode tree, identifying one or more child opcode nodes that depend from the branch opcode; and

implementing an optimized branch node that includes an optimized indexed lookup procedure if such implementation would increase branch processing efficiency and referencing the opcode node from the optimized branch node (pg. 361, last paragraph of section 5, "If there is no gain in using a dynamic access module, the decision tree can by an empty function"),

wherein the indexed lookup procedure is configured to return a set of (key, value) pairs, a single (key, value) pair corresponding to one branch. This last limitation is nothing more than a recitation of steps taken by a hash algorithm, or related types of indexing methods (Applicant's spec. pg. 22, lines 3-7). As such, it would have been obvious to one of ordinary skill in the art at the time of the invention to use an efficient indexing method, when appropriate. For instance, **Graefe et al.** page 360 "using a hash index".

It would have been obvious to one of ordinary skill in the art at the time of the invention to evaluation system of **Aguilera et al.** with the choose plan operators of **Graefe et al.** in order to increase its efficiency as described in **Graefe et al.**

As per claim 11, **Aguilera et al.** as modified, discloses the opcode merger further configured to traverse the opcode tree to determine if an opcode node corresponding to the new query already exists in the opcode tree and add new opcode nodes that correspond to query opcode blocks that are not already included in the opcode tree (**Aguilera et al.,** App. A, pre- processing).

As per claim 12, **Aguilera et al.** as modified, also discloses wherein opcode nodes corresponding to opcode blocks included in a common prefix are represented as a shared segment in the opcode tree (**Aguilera et al.,** App. A).

As per claim 13, **Aguilera et al.** as modified, also discloses wherein queries are merged into the opcode tree dynamically at runtime (**Aguilera et al.,** pg. 2, Col. 1, lines 22-24). It is a well-known goal of subscription server systems, as disclosed by **Aguilera et al.**, to maximize availability. As such, it is anticipated by **Aguilera et al.** that the incremental updates would be done at run-time.

As per claim 14, although **Aguilera et al.** as modified, does not explicitly disclose further comprising Xpath queries in the plurality of queries, it would be obvious to one of ordinary skill in the art to use this system in conjunction with Xpath queries. Evidence for this can be seen by the fact that **Aguilera et al.** has been cited multiple times by papers dealing with XML and Xpath, and listed as related research. The systems are not incompatible, as **Aguilera et al.'s** tests could be tests against Xpath attributes.

As per claim 15, **Aguilera et al.** as modified, discloses the compiler being further configured to create opcode objects that are configured to merge themselves into an appropriate location in the opcode tree (**Aguilera et al.** App A, pre-processing algorithm).

As per claim 17, **Aguilera et al.** as modified, does not explicitly disclose the indexed lookup routine further comprising one of the following routines: a hash routine; a routine that uses tries; an interval tree routine.

However, **Graefe et al.** discloses the indexed lookup routine further comprising one of the following routines: a hash routine; a routine that uses tries; an interval tree routine (**Graefe et al.** pg. 360).

It would obvious to one of ordinary skill in the art at the time of the invention to further modify the **Aguilera et al.** combination to utilize an efficient indexing method based on the attribute in question, as taught by **Graefe et al.**

As per claim 18, **Aguilera et al.** as modified, implicitly has the property that the opcode merger is further configured to restore an optimized branch node to a generic branch node when the optimized branch node is no longer more efficient that the generic branch node (last paragraph of section 5, lines 4-6). The choose-plan is evaluated each time, and causes the most efficient query evaluation technique to be used. If this is the usual sequential method, then it does so.

As per claim 37, **Aguilera et al.** discloses one or more computer-readable media in a physical embodiment, containing computer-executable instructions that, when executed on a computer, perform the following steps:

identifying an opcode block that corresponds to a query to be added to an opcode tree that represents multiple queries with a plurality of opcode nodes (pg. 54, Col. 1, lines 29-35);

identifying an appropriate location in an opcode tree to situate new opcode nodes that correspond to a sequence of opcode objects in the opcode block, the opcode tree including at least one shared opcode node that corresponds to at least two of the multiple queries (Pg. 54, Col. 1, lines 22-24);

evaluating a location context; and

modifying an opcode node or a branch node to incorporate a new opcode node (Pg. 54, Col. 1, lines 22-24).

However, **Aguilera et al.** does not explicitly disclose

wherein, the evaluation step further comprises evaluating a plurality of dependent opcode nodes that depend from a branch node from which the new opcode node depend; and

the modifying step further comprises modifying the branch node to include an indexed lookup function if the dependent opcode nodes perform a similar function and processing the dependent opcode with the indexed lookup function increases the efficiency thereof,

wherein the indexed lookup function is configured to return a set of (key, value) pairs, a single (key, value), pair corresponding to one branch.

However, **Graefe et al.** discloses

wherein, the evaluation step further comprises evaluating a plurality of dependent opcode nodes that depend from a branch node from which the new opcode node depend (pg. 361, last paragraph of section 5, "If there is no gain in using a dynamic access module, the decision tree can by an empty function"); and

the modifying step further comprises modifying the branch node to include an indexed lookup function if the dependent opcode nodes perform a similar function and processing the dependent opcode with the indexed lookup function increases the efficiency thereof (pg. 361, last paragraph of section 5, "If there is no gain in using a dynamic access module, the decision tree can by an empty function"),

wherein the indexed lookup function is configured to return a set of (key, value) pairs, a single (key, value), pair corresponding to one branch. This last limitation is nothing more than a recitation of the steps taken by a hash algorithm, or related types of indexing methods (Applicant's spec. pg. 22, lines 3-7).

It would have been obvious to one of ordinary skill in the art at the time of the invention to modify evaluation system of **Aguilera et al.** with the choose plan operators of **Graefe et al.** in order to increase its efficiency as described in **Graefe et al.**

As per claim 39, this claim is substantially the same as claim 17 above, and is rejected on the same grounds.

As per claim 41, this claim is substantially the same as claim 10 above, and is rejected on the same grounds.

8. **Claims 5, 19-21, 23-28, 30, 35-36, 40, 42-48 are rejected under 35 U.S.C. 103(a) as being unpatentable over Aguilera et al. (NPL #1, Matching Events in a Content-based Subscription System), in view of Graefe et al. (NPL #2, Dynamic Query Evaluation Plans) as applied to claims 1-4, 7-13, 15, 17, 37, 39, 41 above, and in further view of Java API (NPL #3, excerpt from Java 2 Platform Std. Ed. V1.4.2).**

As per claim 5 **Aguilera et al.** as modified, does not explicitly disclose receiving a request to remove a first query from the opcode tree; identifying one or more opcode nodes in the opcode tree that correspond to the first query; removing any identified opcode node that does not correspond to a second query.

However, the **Java API** discloses methods for both inserting and removing nodes (**Java API:** method summary).

If utilized by **Aguilera et al.** as modified, the system would implicitly perform the steps of: receiving a request to remove a first query from the opcode tree; identifying one or more opcode nodes in the opcode tree that correspond to the first query; removing any identified opcode node that does not correspond to a second query (**Aguilera et al.** App. A, **Java API,** method summary).

It would have been obvious to one of ordinary skill in the art at the time of the invention to provide a method of removal similar to the method of insertion as taught by the **Java API** to the **Aguilera et al.** combination in order to maintain the disclosed

space complexity. It is disclosed that is linear with respect to the number of subscriptions (Sec. 4, heading “Space Complexity”).

As per claim 19, **Aguilera et al.** teaches a compiler stored on one or more computer-readable media, in a physical embodiment, containing computer-executable instructions for performing the following steps:

receiving a query to be added to an opcode tree that represents a plurality of queries, at least two of which include similar prefixes (Appendix A, pg 60, predicates are equivalent to opcodes);

compiling a query to produce one or more opcode objects (Sec. 3, ¶ 2, “each subscription is a conjunction of *elementary predicates*);

determininq a function that the opcode object performs (Appendix A, pg 8, ¶ 3);

However, **Aguilera et al.** does not explicitly disclose

determining if a branch node that will reference the opcode node corresponding to the opcode object also references other opcode nodes that perform a similar function; and

implementing an optimized branching function in the branch node including an optimized lookup procedure if the branch node can be optimized to more efficiently process the opcode nodes that it references;

wherein the indexed lookup procedure is configured to return a set of (key, value) pairs, a single (key, value) pair corresponding to one branch.

However, **Graefe et al.** teaches determining if a branch node that will reference the opcode node corresponding to the opcode object also references other opcode nodes that perform a similar function (**Graefe et al.:** pg. 361, last paragraph of section 5, "If there is no gain in using a dynamic access module, the decision tree can by an empty function"); and

implementing an optimized branching function in the branch node including an optimized lookup procedure if the branch node can be optimized to more efficiently process the opcode nodes that it references (**Graefe et al.:** pg. 361, last paragraph of section 5, "If there is no gain in using a dynamic access module, the decision tree can by an empty function"),

wherein the indexed lookup procedure is configured to return a set of (key, value) pairs, a single (key, value) pair corresponding to one branch (This limitation is nothing more than a recitation of the steps taken by a hash algorithm, or related types of indexing methods, Applicant's spec. pg. 22, lines 3-7. As such, it would have been obvious to one of ordinary skill in the art at the time of the invention to use an efficient indexing method, when appropriate. For instance, **Graefe et al.** page 360 "using a hash index").

It would have been obvious to one of ordinary skill in the art at the time of the invention to modify evaluation system of **Aguilera et al.** with the choose plan operators of **Graefe et al.** in order to increase its efficiency as described in **Graefe et al.**

Aguilera et al. as modified, still does not explicitly teach opcode objects that are each configured to merge into the opcode tree as an opcode node by determining an

appropriate location in the tree to merge, and merging into the tree in accordance with a node context of the appropriate location.

Aguilera et al. as modified, techniques are language independent (the exemplary implementation's are not, however, the techniques are), and considering that XML and Xpath were standard in usage at the time of the invention, it would have been obvious to one of ordinary skill to use XML and Xpath to implement these functions.

The **Java API** teaches an extensible Node interface that includes functions that if extended could be used to provide opcode objects that are each configured to merge into the opcode tree as an opcode node by determining an appropriate location in the tree to merge, and merging into the tree in accordance with a node context of the appropriate location. Java Interfaces are designed to provide basic functions, and then be extended for usage. The **Java API** specifically states that it to be used with a Document Object Model, which is used with XML and Xpath. It would be obvious for one of ordinary skill in the art to include language to allow these Java Nodes to insert themselves into a query tree. Numerous tree insertion, sorting, and removal algorithms are well known in the art.

It would be obvious to one of ordinary skill in the art at the time of the invention to combine the teachings of **Aguilera et al.** as modified, to include the teachings of **Java API** because they belong to the same art of efficient query evaluation.

As per claim 20, **Aguilera et al.** as modified, teaches opcode objects that are further configured to merge into the opcode tree only if an identical opcode object

corresponding to a similar query prefix is not already included in the opcode tree (**Aguilera et al.** App. A).

This functionality is implicit if the nodes as provided by claim 19 are used with the matching tree of **Aguilera et al**. The system would not provide the increased efficiency intended by **Aguilera et al.** if it allowed duplication of nodes.

As per claim 21, **Aguilera et al.** as modfiied, teaches it would have been obvious to one of ordinary skill in the art to use the Xpath language in the implementation of this system as Xpath was considered to be the standard at the time of the invention. Important factors when deciding implementation details, such as language, are interoperability and future usage. Using a language and specification that are standard increases the interoperability and lifespan of programs (See **IEEE: Benefits of Standards**, NPL #7).

As per claim 23, **Aguilera et al.** as modified, does not explicitly disclose an optimized branching function further comprises a function selected from the following list of functions: a hash function, an interval tree function, and a function utilizing tries.

. However, **Graefe et al.** discloses an optimized branching function further comprises a function selected from the following list of functions: a hash function, an interval tree function, and a function utilizing tries (**Graefe et al.** pg. 360).

It would obvious to one of ordinary skill in the art at the time of the invention to further modify the **Aguilera et al.** combination to utilize an efficient indexing method based on the attribute in question, as taught by **Graefe et al.**

As per claim 24, **Aguilera et al.** as modified, implicitly has the property wherein the branch node is configured to recognize a context where the optimized branching function is no longer efficient and to resort to its previous function if such a context develops (last paragraph of section 5, lines 4-6). The choose-plan is evaluated each time, and causes the most efficient query evaluation technique to be used. If this is the usual sequential method, then it does so.

As per claim 25, **Aguilera et al.** as modified, discloses the compiler is configured to receive the query and generate the opcode at runtime (**Aguilera et al.** pg. 54, Col. 1, lines 22-24); and the opcode node is configured to merge itself into the opcode tree at runtime (pg. 54, Col. 1, lines 22-24). It is a well-known goal of subscription server systems, as disclosed by **Aguilera et al.**, to maximize availability. As such, it is anticipated by **Aguilera et al.** that the incremental updates would be done at run-time. In addition, moving portions of this functionality to the opcode nodes does not make the system unable to stay active while updating.

As per claim 26, **Aguilera et al.** discloses an opcode object stored on one or more computer-readable media, in a physical embodiment, including computer-executable instructions that, when executed on a computer, perform the following steps:

evaluating a node context of the location to which the new opcode node will be added (**Aguilera et al.** App A, pre-processing algorithm); and

merging itself into the opcode tree by adding and/or modifying references from an opcode node or a branch node to the new opcode node (**Aguilera et al.** App A, pre-processing algorithm),

wherein evaluating, a node context further comprises:

identifying a generic branch opcode from which the new node will depend (Appendix A, pg 8);

identifying one or more other nodes that depend from the generic branch opcode that include a similar expression as the new node (Appendix A, pg 8);

However, **Aguilera et al.** does not explicitly disclose

determining an appropriate location to merge itself as a new opcode node in an opcode tree when a query from which the opcode object is derived is added to a filter table represented by the opcode tree including opcode nodes that, when executed, evaluate the queries;

if a sufficient number of the one or more other nodes exists, modifying the generic branch opcode to an optimized branch opcode that includes an indexed lookup procedure that is optimized to more efficiently process the similar expressions,

wherein the indexed lookup procedure is configured to return a set of i (key, value) pairs, a single (key, value) pair corresponding to one branch.

However, **Graefe et al.** discloses determining an appropriate location to merge itself as a new opcode node in an opcode tree when a query from which the opcode object is derived is added to a filter table represented by the opcode tree including opcode nodes that, when executed, evaluate the queries (**Graefe et al.:** pg. 361, last paragraph of section 5, "If there is no gain in using a dynamic access module, the decision tree can by an empty function");

if a sufficient number of the one or more other nodes exists, modifying the generic branch opcode to an optimized branch opcode that includes an indexed lookup procedure that is optimized to more efficiently process the similar expressions (**Graefe et al.:** pg. 361, last paragraph of section 5, "If there is no gain in using a dynamic access module, the decision tree can by an empty function"),

wherein the indexed lookup procedure is configured to return a set of i (key, value) pairs, a single (key, value) pair corresponding to one branch (This limitation is nothing more than a recitation of the steps taken by a hash algorithm, or related types of indexing methods, Applicant's spec. pg. 22, lines 3-7. As such, it would have been obvious to one of ordinary skill in the art at the time of the invention to use an efficient indexing method, when appropriate. For instance, **Graefe et al.** page 360 "using a hash index").

It would have been obvious to one of ordinary skill in the art at the time of the invention to evaluation system of **Aguilera et al.** with the choose plan operators of **Graefe et al.** in order to increase its efficiency as described in **Graefe et al.**

Aguilera et al. as modified, still does not explicitly teach opcode objects that are each configured to merge into the opcode tree as an opcode node by determining an appropriate location in the tree to merge, and merging into the tree in accordance with a node context of the appropriate location.

The **Java API** teaches an extensible Node interface that includes functions that if extended could be used to provide opcode objects that are each configured to merge into the opcode tree as an opcode node by determining an appropriate location in the tree to merge, and merging into the tree in accordance with a node context of the appropriate location. Java Interfaces are designed to provide basic functions, and then be extended for usage. The **Java API** specifically states that it to be used with a Document Object Model, which is used with XML and Xpath. It would be obvious for one of ordinary skill in the art to include language to allow these Java Nodes to insert themselves into a query tree. Numerous tree insertion, sorting, and removal algorithms are well known in the art.

It would be obvious to one of ordinary skill in the art at the time of the invention to combine the teachings of **Aguilera et al.** as modified, to include the teachings of **Java API** because they belong to the same art of efficient query evaluation.

As per claim 27, **Aguilera et al.** as modified, discloses perform[ing] the recited steps dynamically at runtime (pg. 2, Col. 1, lines 22-24). It is a well-known goal of subscription server systems, as disclosed by **Aguilera et al.**, to maximize availability. As such, it is obvious in view of the teachings of **Aguilera et al.** that the incremental updates would be done at run-time. In addition, moving portions of this functionality to the opcode nodes does not make the system unable to stay active while updating.

As per claim 28, **Aguilera et al.,** does not explicitly disclose performing the recited steps within a .NET environment.

However, it would have been obvious to one of ordinary skill in the art to implement these functions within the .NET framework, given the teachings of the **Java API**. In addition, Java and .NET are considered to be alternatives to each other, both being class libraries designed to run on a virtual machine. Therefore, it is considered an obvious implementation detail to choose either Java or .NET based on the developer's familiarity.

As per claim 30, the combination of **Aguilera et al.** as modified, teaches wherein evaluating a node context further comprises:

identifying an optimized branch opcode from which the new node will depend (Appendix A, pg 8);

identifying one or more other nodes that depend from the optimized branch opcode that include a similar expression as the new node (Appendix A, pg 8); and

if minimum threshold number of the one or more other nodes is not met, modifying the optimized branch opcode to a generic branch opcode that can process the number of one or more other nodes more efficiently than the optimized branch opcode can (**Graefe et al.:** pg. 361, last paragraph of section 5, "If there is no gain in using a dynamic access module, the decision tree can by an empty function").

Graefe et al. teaches a dynamic optimization technique that can be placed at any point in a query evaluation plan, such as a branch node input to hold the optimization algorithm. The decision to either add or remove an optimization technique is provided by **Graefe et al.**, and would be decided upon at the same time.

As per claim 31, although **Aguilera et al.** does not explicitly disclose receiving a request to remove a first query from the opcode tree; identifying one or more opcode nodes in the opcode tree that correspond to the first query; removing any identified opcode node that does not correspond to a second query.

However, **Aguilera et al.** does state that the disclosed algorithm has space complexity that is linear with respect to the number of subscriptions (Sec. 4, heading "Space Complexity"). As such, it would have been obvious to one of ordinary skill in the art at the time of the invention to include a method of removing expired queries. As **Aguilera et al.** is silent on the point of subscription (query) removal, one of ordinary skill in the art would know to execute the steps described in claim 5, in order to remove expired queries, and maintain the minimum size of the tree. This is further shown in the **Java API**. The Node Interface has methods to facilitate removal of nodes.

In addition, **Aguilera et al**. inherently teaches the step of modifying a branch node that references an opcode node that is removed from the opcode tree. This is considered inherent in the system of claim 31, as it would fail if it did not provide this step, with the minimal requirement of removing the reference to the removed node.

In addition, it would have been obvious to one of ordinary skill in the art at the time of the invention to include the step of removing an optimized lookup function that includes an indexed lookup routine from the branch node if removing the branch node renders the lookup function less efficient than a direct comparison function (**Graefe et al.:** pg. 361, last paragraph of section 5, "If there is no gain in using a dynamic access module, the decision tree can by an empty function"),

wherein the indexed lookup routine is configured to return a set of (key, value) pairs, a single (key, value) pair corresponding to one branch (This limitation is nothing more than a recitation of the steps taken by a hash algorithm, or related types of indexing methods, Applicant's spec. pg. 22, lines 3-7. As such, it would have been obvious to one of ordinary skill in the art at the time of the invention to use an efficient indexing method, when appropriate. For instance, **Graefe et al.** page 360 "using a hash index").

As per claim 33, **Aguilera et al.** as modified does not explicitly teach the modifying further comprises removing the branch node if the branch node references only one other opcode node other than the opcode node to be removed.

However, it would have been obvious to one of ordinary skill in the art at the time of the invention that the modifying further comprises removing the branch node if the branch node references only one other opcode node other than the opcode node to be removed. This is considered obvious because having a branch node where the tree no longer branches is superfluous and would waste unneeded time and space.

As per claim 35, **Aguilera et al.** as modified teaches the modifying further comprises implementing an optimized processing function in the branch node if the removal of the branch node creates a context in which the optimized processing function would increase efficiency of the branch node processing (**Graefe et al.:** pg. 361, last paragraph of section 5, "If there is no gain in using a dynamic access module, the decision tree can by an empty function").

As per claim 36, this claim is substantially the same as claim 23 above, and is rejected on the same grounds.

As per claim 40, this claim is substantially the same as claim 21 above, and is rejected on the same grounds.

As per claim 42, this claim is substantially the same as claim 19 above, and is rejected on the same grounds.

As per claim 43, **Aguilera et al.** as modified, does not explicitly disclose wherein the steps are performed in a Common Language Runtime (CLR) environment. It would have been obvious to one of ordinary skill in the art at the time of the invention that CLR is an execution environment for the .NET library, and is an alternative to the Java Virtual Machine, which Java runs on. Therefore, it would have been obvious to one of ordinary skill in the art at the time of the invention to perform the steps in a Common Language Runtime (CLR) environment, if they were implementing the invention in a .NET framework rather than Java. Choice of implementation language is not considered a patentable distinction

As per claim 44, **Aguilera et al.** as modified, discloses wherein for each given (key, value) pair corresponding to one branch, the key is the value of matching literals in the branch, and the value identifies the branch to which the literal belongs (**Graefe et al.,** pg 360). These steps are implicit for making use of a hash table, or other efficient lookup index. See applicant spec. pg. 22, lines 3-7, "the value of the literal is *hashed* to derive an index entry" (emphasis added), in particular.

As per claim 45, this claim is substantially the same as claim 44 above, and is rejected on the same grounds.

As per claim 46, this claim is substantially the same as claim 44 above, and is rejected on the same grounds.

As per claim 47, this claim is substantially the same as claim 44 above, and is rejected on the same grounds.

As per claim 48, this claim is substantially the same as claim 44 above, and is rejected on the same grounds.

Conclusion

9. Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

Any inquiry concerning this communication or earlier communications from the examiner should be directed to AARON CHILES whose telephone number is (571)270-

3424. The examiner can normally be reached on Monday-Thursday, Noon - 6:00 p.m., EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Mohammad Ali can be reached on (571) 272-4105. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

/J. F. B./
Examiner, Art Unit 2164

ac

/Mohammad Ali/
Supervisory Patent Examiner, Art Unit 2169